Cranfield UNIVERSITY

Applications in Practical High-End Computing - Group Project

Assignment - "Workflow"

Implementation document

Supervisor: Dr Stuart Barnes

Authors: Mateusz Gołąb
Csaba Kerti
Jakub Kiełbasa
Zsolt Kollarits

Course: CSTE / SETC

Table of contents

1. Introduction

After taking into consideration all advantages and disadvantages we decided to use JAVA language for implementation and SSH protocol for communication with remote server. In this document we can see an overview about these technologies.

2. Overview of JAVA

a) Why JAVA?

Java is a standard Object Oriented Programming Language and contains all of its benefits: [1]

- supports the construction of programs that consist of collections of collaborating objects
- encapsulation, inheritance, polymorphism
- multithreaded
- robust and secure

But this attributes can be found all modern programming language, so why JAVA? The reason is the **multiplatform** usability. Ideally, this means Java can be developed on any device, compiled into a standard *bytecode* and be expected to run on any device equipped with a *Java .virtual machine* (JVM). [2]

1. Figure: JAVA multiplatform [3]

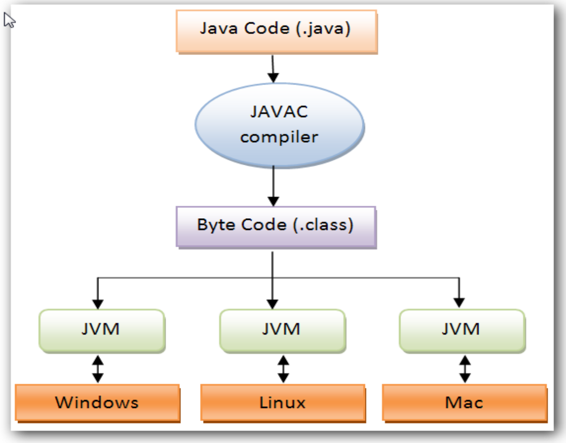
b) Compile process:

Java compiler will not use the variables and methods for the numerical compiler invoked not sure procedures in the implementation of the memory layout, but these symbols will be used to retain information in the byte code, the explanation for the operation process of creating memory layout, and then look-up table to determine a method of address. This effectively guaranteed the Java portability and security. [4]

c) About JVM:

JVM is designed to provide a description of specifications based on an abstract computer model for the interpretation of the procedures developers provide excellent flexibility, but also ensure that Java code can be in conformity with the norms of running on any system. JVM certain aspects of its realization are given a specific definition, particularly on Java executable code that *bytecode* provided a clear format specifications. The specifications including operation codes and the number of operators and numerical syntax, the numerical identifier that, as well as the Java class files in Java objects, constants in the JVM buffer pool storage mapping. These definitions for the JVM interpreter developers with the necessary information and development environment. [4]

Java interpreter running Java byte code equivalent to the "CPU", but the "CPU" is not achieved through hardware, but software to achieve. [4]



2. *Figure*: JAVA execution process (and multiplatform usability) [5]

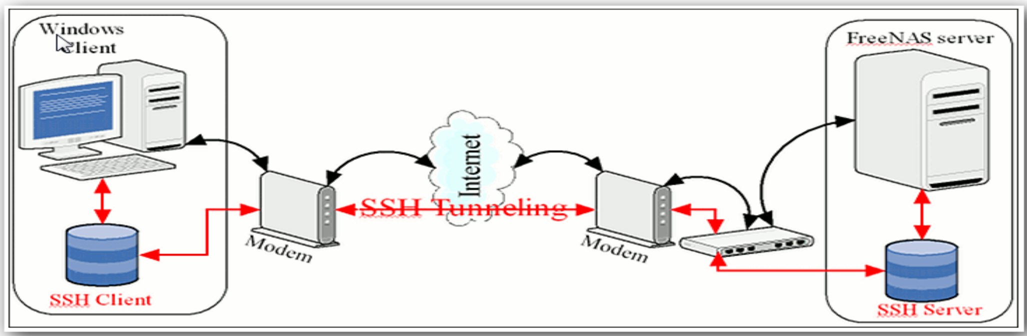
3. Communication with remote server – SSH

The program uses the *SSH* protocol to communicate with the server. This method is widely used, popular technique.

But the most important attribute of SSH (why we have chosen it) the universal usage. What does it mean? SSH is **platform-independent**, can be used for many applications across many platforms (i.e. Windows, Linux, Mac).
The SSH *Server* runs on UNIX, Linux and VAX. *Client* runs on the above, plus Windows and many other platforms. [6]
Some of the applications below may require features that are only available or compatible with specific SSH clients or servers. [7] The **multiplatform** behaviour is a very important attribute or system can be more flexible with SSH.

a) About SSH:

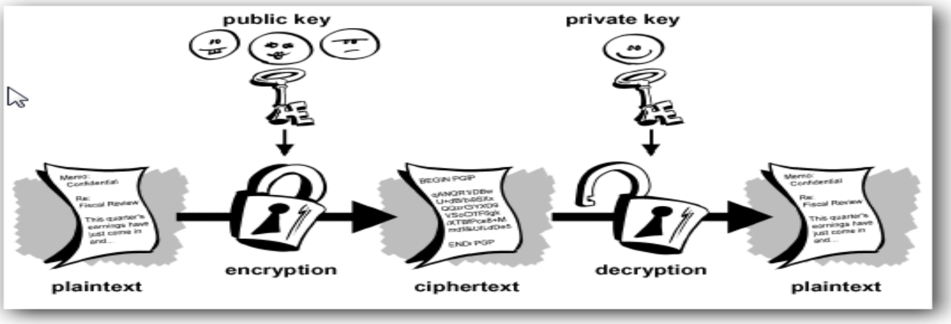
Secure Shell (SSH) is a network protocol for secure data communication, remote shell services or command execution and other secure network services between two networked computers that it connects via a secure channel over an insecure network: a server and a client (running SSH server and SSH client programs, respectively). The protocol specification distinguishes two major versions that are referred to as SSH-1 and SSH-2. [7]



3. *Figure*: SSH platform-independent model [8]

b) Security of SSH:

The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network, such as the Internet. SSH uses public-key cryptography to authenticate the remote computer and allow it to authenticate the user, if necessary. Anyone can produce a matching pair of different keys (public and private). The public key is placed on all computers that must allow access to the owner of the matching private key (the owner keeps the private key secret). While authentication is based on the private key, the key itself is never transferred through the network during authentication. SSH only verifies if the same person offering the public key also owns the matching private key. [7]



4. Figure: SSH encryption process [9]

4. Source code examples

a) WorkflowManager interface:

```
public interface WorkflowManager extends Observer
{

    public void startSimulation(WorkflowSequence sequence);

    public void sendResult();
}
```

b) WorkflowQueue interface:

```
public interface WorkflowQueue
{

      public boolean isEmpty();

      public Simulation pop();

      public void push(Simulation simulation);
}
```

c) WorkflowSequence interface:

```
public interface WorkflowSequence
{

    public int getNumberOfModules();

    public void addModule(Module module);

    public void removeModule(Module module);

    public void executeModule();

    public void nextModule();

    public void validateModuleInput();

    public void validateModuleOutput();

    public void startSimulation(Simulation simulation);

    public StablePoint createStablePoint();

    public Boolean isOutputStateCorrect();

    public Boolean isInputStateCorrect();

    public WorkflowSequenceState getState();

    public void recoverFromStablePoint(StablePoint stablePoint);

    public Module getCurrentModule();
}
```

d) WorkflowManager Update function:

```
@Override
public void update(Observable o, Object arg)
{
    if (o instanceof WorkflowSequence)
    {
        WorkflowSequence sequence = (WorkflowSequenceImpl) o;
        switch (sequence.getState()) {

            case MODULE_INPUT_VALIDATION:
                if (sequence.isInputStateCorrect())
                {
                    view.printInputValidated();
                    sequence.executeModule();
                }
                else
                {
                    view.printError("Module input is incorrect");
                    recover(sequence);
                }

                break;

            case MODULE_OUTPUT_VALIDATION:
                if (sequence.isOutputStateCorrect())
                {
                    view.printOutputValidated();

                view.printModuleFinished(sequence.getCurrentModule().to
                String());
                    sequence.nextModule();
                }
                else
                {
                    view.printError("Module output is incorrect");
                    recover(sequence);
                }

                break;

            case ERROR:
                view.printError("Module error");
                recover(sequence);
                break;

            case START_NEW_SIMULATION:
                startSimulation(sequence);
                break;

            case SIMULATION_RECOVERY_RESTART:
                restart(sequence);
                break;
            case SIMULATION_SUCCESS:
                result = true;
                view.printSimulationSuccessfullyFinished();
                sendResult();
                break;
            case SIMULATION_FAILURE:
                result = false;
                view.printSimulationUnsuccessfullyFinished();
                sendResult();
                break;
        }
    }
}
```

e) WorkflowManager Recovery function:

```
private void recover(WorkflowSequence sequence)
{
        // get stable point
        StablePoint stablePoint = null;
        if (lastBackupPerformed < 2)
        {
            stablePoint = database.getLastStablePoint();
            lastBackupPerformed++;
        }
        else
        {
            stablePoint = database.getPreviousStablePoint();
            backupsPerformed++;
            }
            // recover using stable point
            if (stablePoint != null)
            {
                sequence.recoverFromStablePoint(stablePoint);
            }
            else
            {
                view.printError("Recovery  finished  ,  simulation  could  not
finish successfully");

                // current simulation failed, starting new simulation
                startSimulation(sequence);
            }
  }
```

f) WorkflowQueue implementation:

```
public class WorkflowQueueImpl implements WorkflowQueue
{
    private LinkedList<Simulation> queue;
    public WorkflowQueueImpl()
    {
        queue = new LinkedList<Simulation>();
    }
    @Override
    public boolean isEmpty()
    {
        return queue.isEmpty();
    }
    @Override
    public Simulation pop()
    {
        return queue.removeFirst();
    }
    @Override
    public void push(Simulation simulation)
    {
        queue.addLast(simulation);
    }
}
```

g) WorkflowSequence add/remove module functions:

```
@Override
public void addModule(Module module)
{
    if (modules.isEmpty())
    {
        modules.add(module);
        currentModule = modules.get(0);
    }
```

```
        else
        {
            modules.add(module);
        }
    }

    @Override
    public void removeModule(Module module)
    {
        if (currentModule.equals(module))
        {
            if (modules.indexOf(module) == modules.size() - 1)
            {
                currentModule = modules.get(0);
            }
            else
            {
                // iterator.remove();
                currentModule  =  modules.get(modules.indexOf(currentModule)
+ 1);
            }
        }
        modules.remove(module);
    }
```

h) WorkflowSequence execute module function:

```
public void executeModule()
    {
        if (currentModule.execute() == false)
        {
            state = WorkflowSequenceState.ERROR;
            notifyObserver();
        }
        else
        {
            validateModuleOutput();
        }
    }
```

i) WorkflowSequence create stable point to database:

```
@Override
public StablePoint createStablePoint()
{
    int prev = -1;
    int next = -1;

    int index = modules.indexOf(currentModule);

    if (index > 0)
        prev = modules.get(index - 1).getID();
```

```
            if (index < modules.size() - 1)
                next = modules.get(index + 1).getID();

            return   new   StablePoint(prev,   next,   "previousOutputFilePath",
    "nextInputFilePath", iterationNumber);
        }
```

j) WorkflowSequence get next module function:

```
@Override
public Module nextModule()
{
    int index = modules.indexOf(currentModule);
    if (index == modules.size() - 1)
    {
        iteration++;
        if (iteration >= iterationNumber)
        {
            state = WorkflowSequenceState.SIMULATION_SUCCESS;
            notifyObserver();

            return null;
        }

        view.printSimulation(iteration);
        currentModule = modules.get(0);
    }
    else
    {
         currentModule = modules.get(index + 1);
    }
    validateModuleInput();

    return currentModule;
}
```

k) WorkflowSequence recovery a stable point function:

```
@Override
public void recoverFromStablePoint(StablePoint stablePoint)
{
    for (Module m : modules)
    {
        if (m.getID().equals(stablePoint.getPreModuleID()))
        {
            currentModule = m;
            validateModuleInput();
            return;
        }
    }
}
```

5. Examples of output

a) Workflow executed successfully:

```
Simulation started.
Module's input(s) validated successfully.
Module's output(s) validated successfully.
1 module finished.
Module's input(s) validated successfully.
Module's output(s) validated successfully.
2 module finished.
Module's input(s) validated successfully.
Module's output(s) validated successfully.
3 module finished.
Module's input(s) validated successfully.
Module's output(s) validated successfully.
4 module finished.
Simulation process successfully finished
```

b) Workflow failed, recovery succeed:

```
Simulation started.
Error occured in WorkflowManager! Type: Module input is incorrect.
Error occured in WorkflowManager! Type: Module input is incorrect.
Error occured in WorkflowManager! Type: Module input is incorrect.
Error  occured  in  WorkflowManager!  Type:  Recovery  finished  ,
simulation could not finish successfully.
Simulation proess failure! Reason: WorkflowQueue is EMPTY.
```

6. References

[1]
http://it.toolbox.com/blogs/enterprise-solutions/key-features-of-the-java-language-41397

[2]
http://en.wikipedia.org/wiki/Write_once,_run_anywhere

[3]
http://erpscan.com/wp-content/uploads/2011/05/09.png

[4]
http://www.anyang-window.com.cn/java-cross-platform-principles/

[5]
http://viralpatel.net/blogs/2008/12/java-virtual-machine-an-inside-story.html

[6]
http://boran.ch/security/sp/ssh-part1.html

[7]
http://en.wikipedia.org/wiki/Secure_Shell

[8].
http://wiki.freenas.org/documentation:how_to_access_the_webgui_via_ssh_tunnel_from_a_windows_shortcut

[9]
http://4.bp.blogspot.com/-VpNda7dWLpU/TmYn7PiFUVI/AAAAAAAAAQ8/FJmkiWEKxcE/s640/public_key_encryption.gif